Anton, a Special-Purpose Machine for Molecular Dynamics Simulation

David E. Shaw, Martin M. Deneroff, Ron O. Dror, Jeffrey S. Kuskin, Richard H. Larson, John K. Salmon, Cliff Young, Brannon Batson, Kevin J. Bowers, Jack C. Chao, Michael P. Eastwood, Joseph Gagliardo, J.P. Grossman, C. Richard Ho, Douglas J. Ierardi, István Kolossváry, John L. Klepeis, Timothy Layman, Christine McLeavey, Mark A. Moraes, Rolf Mueller, Edward C. Priest, Yibing Shan, Jochen Spengler, Michael Theobald, Brian Towles, and Stanley C. Wang

D. E. Shaw Research, LLC, New York, NY 10036, USA

ABSTRACT

The ability to perform long, accurate molecular dynamics (MD) simulations involving proteins and other biological macromolecules could in principle provide answers to some of the most important currently outstanding questions in the fields of biology, chemistry and medicine. A wide range of biologically interesting phenomena, however, occur over time scales on the order of a millisecond—about three orders of magnitude beyond the duration of the longest current MD simulations.

In this paper, we describe a massively parallel machine called Anton, which should be capable of executing millisecond-scale classical MD simulations of such biomolecular systems. The machine, which is scheduled for completion by the end of 2008, is based on 512 identical MD-specific ASICs that interact in a tightly coupled manner using a specialized high-speed communication network. Anton has been designed to use both novel parallel algorithms and special-purpose logic to dramatically accelerate those calculations that dominate the time required for a typical MD simulation. The remainder of the simulation algorithm is executed by a programmable portion of each chip that achieves a substantial degree of parallelism while preserving the flexibility necessary to accommodate anticipated advances in physical models and simulation methods.

Categories and Subject Descriptors: J.3 [*Computer Applications*]: Life and Medical Sciences – Biology and genetics; Health; C.1.4 [*Computer Systems Organization*]: Processor Architectures – parallel architectures; C.3 [*Computer Systems Organization*]: Special-purpose and Application-based Systems – microprocessor/microcomputer applications

General Terms: Algorithms; Performance; Design.

David E. Shaw is also with the Center for Computational Biology and Bioinformatics, Columbia University, New York, NY 10032. E-mail correspondence: david@deshaw.com.

ISCA'07, June 9–13, 2007, San Diego, California, USA.

Keywords: Molecular dynamics; Special-purpose machine; Biomolecular system simulation; Computational biology; Protein structure; Protein folding; Computational drug design; Bioinformatics.

1. INTRODUCTION

Molecular dynamics (MD) simulations can be used to model the motions of molecular systems, including proteins, cell membranes and DNA, at an atomic level of detail. A sufficiently long and accurate MD simulation could allow scientists and drug designers to visualize for the first time many critically important biochemical phenomena that cannot currently be observed in laboratory experiments, including the "folding" of proteins into their native three-dimensional structures, the structural changes that underlie protein function, and the interactions between two proteins or between a protein and a candidate drug molecule [8, 14, 18, 26]. Such simulations could answer some of the most important open questions in the fields of biology and chemistry, and have the potential to make substantial contributions to the process of drug development.

Many of the most important biological processes occur over time scales on the order of a millisecond. MD simulations on this time scale, however, lie several orders of magnitude beyond the reach of current technology; only a few MD runs have thus far reached even a microsecond of simulated time, and the vast majority have been limited to the nanosecond time scale [9, 27]. Millisecond-scale simulations of a biomolecular system containing tens of thousands of atoms will in practice require that the forces exerted by all atoms on all other atoms be calculated in just a few microseconds—a process that must be repeated on the order of 10^{12} times. Such simulations will require the use of a huge number of arithmetic processing elements, and because of the global nature of the inter-atomic force calculations, will involve a great deal of inter-processor communication, which must be carefully managed in order to preserve scalability. These stringent requirements far exceed the current capabilities of even the most powerful commodity clusters or general-purpose scientific supercomputers.

In this paper, we describe a specialized, massively parallel machine, called Anton, that is designed to accelerate MD simulations by several orders of magnitude, bringing millisecond-scale simulations within reach for molecular systems involving tens of thousands of atoms. The machine, which is scheduled for com-

pletion by the end of 2008, will comprise 512 processing nodes in its initial configuration, each containing a specialized MD computation engine implemented as a single ASIC (Section 4). The molecular system to be simulated is decomposed spatially among these processing nodes, which are connected through a specialized high-performance network to form a three-dimensional torus. Anton's expected performance advantage is attributable to a combination of MD-specific hardware that achieves a very high level of arithmetic density and novel parallel algorithms that, among other things, enhance scalability by reducing both intra- and inter-chip communication.

In designing Anton and its associated software, we have attempted to attack a somewhat different problem than the ones addressed by several other projects that have deployed significant computational resources for MD simulations. The Folding@Home project [21], for example, has obtained a number of significant and interesting scientific results by using as many as 200,000 PCs (made available over the Internet by volunteers) to simulate a very large number of *separate* molecular trajectories, each of which is limited to the time scale accessible on a single PC. While a great deal can be learned from a large number of independent MD trajectories, many other important problems require the examination of a single, very long trajectory—the principal task for which Anton is designed. Other projects, such as FASTRUN [13], MDGRAPE [31], and MD Engine [32], have produced special-purpose hardware to accelerate the most computationally expensive elements of an MD simulation. Such hardware reduces the *cost* of MD simulations, particularly for large molecular systems, but Amdahl's law and communication bottlenecks prevent the efficient use of enough such chips in parallel to extend individual simulations beyond microsecond time scales.

Anton is named after Anton van Leeuwenhoek, whose contributions to science and medicine we hope to emulate in our own work. In the 17th century, van Leeuwenhoek, often referred to as the "father of microscopy," built high-precision optical instruments that allowed him to visualize for the first time an entirely new biological world that had previously been unknown to the scientists of his day. Using his primitive microscope, he was the first to see bacteria and other microorganisms (helping to elucidate the origins of infectious disease), and was the discoverer of blood cells and spermatozoa (helping to establish modern theories of respiration and reproduction). We view Anton (the machine) as a sort of "computational microscope." To the extent that we and other researchers are able to increase the length of molecular dynamics simulations, just as van Leeuwenhoek increased the effective magnification of the optical instruments of his day, we would hope to provide contemporary biological and biomedical researchers with a tool for understanding organisms and their diseases on a still smaller length scale.

2. STRUCTURE OF MOLECULAR DYNAMICS COMPUTATIONS

An MD computation simulates the motion of a collection of atoms (the *chemical system*) over a period of time according to the laws of classical physics. The chemical system might consist of a protein and its surrounding environment (*solvent*, usually water), and might also include other types of molecules, such as lipids, carbohydrates, nucleic acids, or drug molecules. The entire chemical system occupies a small parallelepiped (the *global cell*), typically tens of angstroms on a side, filled with tens or hundreds of thousands of atoms. In the interest of accuracy, we represent solvent molecules explicitly, rather than using a continuum approximation. We also typically assume that the global cell tiles an infinite space by repeating in each dimension with a period equal to the side length of the global cell in that dimension (*periodic boundary conditions*).

For brevity, we describe only those details of MD calculations required to explain our architecture. Numerous more complete surveys of MD methodology are available [2, 14, 26]. For expository simplicity, we assume a one-to-one correspondence between particles and atoms. We also assume that the global cell is a *rectangular* parallelepiped, although our hardware also supports non-rectangular global cells.

An MD computation breaks time into a series of discrete *time steps*, each representing a few femtoseconds of simulated time. A time step has two major phases. *Force calculation* computes the force on each particle due to other particles in the system. *Integration* uses the net force on each particle to update that particle's position and velocity. Simulating a millisecond of time requires on the order of 10^{12} time steps.

2.1 Force Calculation

Interatomic forces are calculated based on a *molecular mechanics force field* (or simply *force field*), which models the total potential energy of a chemical system as a relatively simple function of the atomic spatial coordinates. The force on a given particle is then the gradient of the potential energy function with respect to the coordinates of that particle. Although classical MD simulation is inherently an approximation, it is dramatically faster than directly solving the full set of quantum mechanical equations. Several decades of work have gone into the development of biomolecular force fields through fitting models to experimental and quantum data.

Anton is compatible with commonly used biomolecular force fields, including CHARMM [21], AMBER [19], and OPLS-AA [17]. All of these express the total system potential energy E_{total} as the sum of several contributions:

$$E_{\text{total}} = E_{\text{bonded}} + E_{\text{nonbonded}}$$

$$E_{\text{bonded}} = E_{\text{length}} + E_{\text{angle}} + E_{\text{dihedral}}$$

$$E_{\text{nonbonded}} = E_{\text{es}} + E_{\text{vdw.}}$$

The bonded energy term E_{bonded} applies only to atoms that are separated by no more than three covalent bonds. The bonded energy is a sum of three kinds of terms: the length term E_{length} of a bond between two directly bonded atoms; the angle term E_{angle} defined by the positions of three atoms, two of which are bonded to a third; and the dihedral (torsion) angle term E_{dihedral} defined by four atoms connected by three sequential bonds.

The nonbonded energy term $E_{\text{nonbonded}}$ is the sum of an electrostatic component E_{es} and a van der Waals component E_{vdw}. These are known as nonbonded terms because they include interactions between all pairs of atoms in the simulated system with the *exception* of those separated by a small number of covalent bonds. Biomolecules are sparsely connected, typically having no more than four bonds per atom, making the number of bonded

interactions linear in the number of atoms. The number of nonbonded interactions, on the other hand, scales quadratically with the number of atoms, and typically far exceeds the number of bonded terms. Van der Waals forces fall off sufficiently quickly with distance that they can typically be neglected for pairs of particles separated by more than some *cutoff radius*, usually chosen between 5 and 15 Å. Neglecting electrostatic interactions beyond a cutoff, however, leads to serious artifacts in explicit solvent simulations [7, 23]. Electrostatic forces are thus typically computed by one of several efficient, approximate methods that account for long-range interactions without requiring the explicit interaction of all pairs of particles.

Anton uses a method we developed for fast electrostatic calculations called *k-space Gaussian split Ewald*, or *k*-GSE [28]. This algorithm reduces the computational workload of the nonbonded interactions from $O(n^2)$ to $O(n \log n)$ in the number of particles, and maps to our specialized hardware more effectively than previously described methods for efficient electrostatics computation. In *k*-GSE, as in other similar long-range electrostatics algorithms such as PME [10] and PPPM [16], electrostatic interactions are divided into two contributions. The first decays rapidly with particle separation, and is thus computed directly for all particle pairs separated by less than the cutoff radius. We refer to this contribution, together with the range-limited van der Waals interactions, as *explicit pairwise nonbonded interactions*. The second contribution (*long-range interactions*) decays more slowly, but can be expressed as a convolution with a smooth kernel that can be efficiently computed by taking the fast Fourier transform (FFT) of the charge distribution on a regular mesh, multiplying by an appropriate function in Fourier space, and taking an inverse FFT. Charge must be mapped from particles to nearby mesh points before the FFT computation (*charge spreading*), and forces on particles must be calculated from the convolution result at nearby mesh points after the inverse FFT computation (*force interpolation*). In contrast with PME and PPPM, *k*-GSE performs charge spreading and force interpolation using Gaussian interpolation kernels. The spherical symmetry of these Gaussians allows us to use the same hardware for charge spreading and force interpolation as for computing explicit pairwise interactions. (Section 4 describes this hardware in more detail.)

In most force fields, the force between each pair of particles that participate in a length, angle, or dihedral term is either eliminated from or underweighted in the nonbonded force calculation. On Anton, this effect can be achieved most efficiently by computing nonbonded interactions between *all* pairs of particles, then subtracting compensatory *force correction terms*.

2.2 Integration

The integration phase of MD uses the results of force computation to update atomic positions and velocities, numerically integrating a set of ordinary differential equations corresponding to Newton's laws of motion. The numerical integrators used in MD are nontrivial for several reasons. First, the integration algorithm and the manner in which numerical issues are handled can have a significant effect on accuracy. Second, some simulations require the integrator to calculate and adjust global properties like temperature and pressure. Finally, one can significantly accelerate most simulations by incorporating *constraints* that eliminate the fastest vibrational motions. For example, we typically fix the lengths of bonds to all hydrogen atoms and hold water molecules rigid.

2.3 Parallelization

In Anton, as in most parallel MD codes, the global cell is divided into a regular grid of smaller rectangular parallelepipeds, which will be referred to here as *boxes*. Each processing node updates the positions and velocities of particles in one box, referred to as the *home box* of that node and of those particles. Conversely, we refer to each particle as having a *home node*.

To parallelize explicit pairwise nonbonded interactions, our machine uses an algorithm we developed called the *NT method* [29]. The NT method achieves both asymptotic and practical reductions in required interprocessor communication bandwidth relative to traditional parallelization methods. In traditional spatial decompositions [25], the interaction between two particles is always calculated by the home node of one or both particles. The NT method is one of a number of novel methods [5, 6, 11, 15, 29, 30] that also employ a spatial assignment of particles to nodes, but that often compute the interaction between two particles in a node on which neither particle resides, and which must thus import information related to both particles. We refer to such techniques as *neutral territory methods* [5, 6].

In the NT method, each node requires access to the positions of all particles within two regions of space, known as the *tower* and the *plate* of that node. Both of these regions include the node's home box, but also include portions of neighboring boxes. Each node must thus import position data from certain neighboring nodes. Each node then calculates the interaction between each pair of particles, one from its plate and one from its tower, that are separated by no more than the cutoff radius (with the exception of a limited number of pairs that are excluded to avoid duplicate interactions). Because explicit pairwise nonbonded interactions constitute the majority of the computation required in an MD simulation, much of the ASIC's area is devoted to hardware pipelines specialized for these interactions. We also use the NT method, with minor modifications, to parallelize the interaction of particles with *grid points* in the course of charge spreading and force interpolation.

Several other parts of an MD simulation require interprocessor communication. Force computations require the transfer of data for long-range electrostatic calculations (including forward and inverse FFTs) and for the evaluation of bonded forces. Additional communication is also required in the course of integration for the enforcement of constraints, the evaluation of global quantities such as pressure and temperature, and the transfer of particles from one node to another as they move between boxes over the course of a simulation (*atom migration*).

3. WHY SPECIALIZED HARDWARE?

A natural question is whether a specialized machine for molecular simulation can gain a significant performance advantage over general-purpose hardware. After all, history is littered with the corpses of specialized machines, spanning a huge gamut from Lisp machines [20] to database accelerators [3]. Performance and transistor count gains predicted by Moore's law, together with the economies of scale behind the development of commodity processors, have driven a history of general-purpose microprocessors

outpacing special-purpose solutions. Any plan to build specialized hardware must account for the expected exponential growth in the capabilities of general-purpose hardware.

We concluded that special-purpose hardware is warranted in this case because it leads to a much greater improvement in absolute performance than the expected speedup predicted by Moore's law over our development time period, and because we are currently at the cusp of simulating time scales of great biological significance. We expect Anton to run simulations over 1000 times faster than was possible when we began this project. Assuming that transistor densities continue to double every 18 months and that these increases translate into proportionally faster processors and communication links, one would expect approximately a ten-fold improvement in commodity solutions over the five-year development time of our machine (from conceptualization to bringup). We therefore expect that a specialized solution will be able to access biologically critical millisecond time scales significantly sooner than commodity hardware.

Specialization permits us to tailor the system specifically to MD rather than attempting to speed up general-purpose parallel computing. That is, we need not speed up all high-performance codes; we need only speed up MD. This means that "tuning to the application" is not illegal or questionable as it is for general-purpose machines. Quite the opposite is true—as long as we perform the correct MD calculations, application-specific optimizations and hardware structures are valid and encouraged.

To simulate a millisecond within a couple of months, we must complete a time step every few microseconds, or every few thousand clock ticks. The sequential dependence of successive time steps in an MD simulation makes speculation across time steps extremely difficult. Fortunately, specialization offers unique opportunities to accelerate an individual time step using a combination of architectural features that reduce both computational latency and communication latency.

For example, we reduced computational latency by designing:

- Dedicated, specialized hardware datapaths and control logic to evaluate the pairwise explicit nonbonded interactions and to perform charge spreading and force interpolation. In addition to packing much more computational logic on a chip than is typical of general-purpose architectures, these pipelines use customized precision for each operation, increasing the speed of the datapath logic and decreasing the silicon area required per arithmetic unit.
- Specialized, yet programmable, processors to compute bonded interactions and the FFT and to perform integration. The instruction set architecture (ISA) of these processors is tailored to the geometric calculations involved in bonded force calculation and constrained integration, and to the communication patterns involved in MD. Their programmability provides flexibility to accommodate various force fields and integration algorithms.
- Dedicated support in the memory system to accumulate forces for each particle without using the programmable processors.

Table 1. Profile of a single-processor run using the GROMACS MD package [33].

Phase	Task	% execution time
Force Calculation	Explicit pairwise nonbonded interactions	60
	FFT, inverse FFT, & Fourier space multiplication	17
	Charge spreading and force interpolation	13
	Bonded force terms	1
	Correction force terms	4
Integration	Position & velocity updates	2
	Constraints	2
	Pressure computation	1

Anton performs nearly all its calculations in fixed-point arithmetic, which provides greater speed and accuracy in MD calculations than floating-point logic occupying the same silicon area.

We reduced communication latency by designing:

- A low-latency, high-bandwidth network, both within an ASIC and between ASICs, that includes specialized routing support for common MD communication patterns such as multicast and compressed transfers of sparse data structures.
- A system of hardware features that enable choreographed "push"-based communication. Producers send results to consumers without the consumers having to request the data beforehand, and the consumers have counters that allow them to detect when all required data have arrived.
- Synchronization support at various points in the ASIC that allow each point to begin computation when all necessary data have arrived, without the need to receive a "start" indication from a central coordinator.
- A set of autonomous direct memory access (DMA) engines that offload communication tasks from the computational units, allowing greater overlap of communication and computation.
- Admission control features that prioritize packets carrying certain algorithm-specific data types.

We balance our design very differently from a general-purpose supercomputer architecture. Relative to other high-performance computing applications, MD uses much communication and computation but surprisingly little memory. Consider an MD simulation of 25,000 particles. If each particle requires 64 bytes of storage, then the entire architectural state is just 1.6 MB. Divided among the 512 nodes of a typical machine, this is only 3.2 KB per node, which would fit handily into the L1 cache of any modern processor. We exploit this property by using only SRAMs and small L1 caches on our ASIC, with all code and data fitting on-chip in normal operation. Rather than spending silicon area on large caches and aggressive memory hierarchies, we instead dedicate it to communication and computation.

It is serendipitous that the most computationally intensive parts of MD—in particular, the electrostatic interactions—are also the most well-established and unlikely to change as force field models evolve, making them particularly amenable to hardware acceleration. Dramatically accelerating MD simulation, however, requires that we accelerate more than just an "inner loop." Table 1 shows that explicit pairwise nonbonded computations account for 60% of the computational time for a particular MD simulation on a single general-purpose processor. Long-range interactions, including charge spreading, force interpolation, and FFT computation, account for another 30% of the time, bringing the total fraction of time spent on nonbonded computations to 90%. Amdahl's law states that no matter how much we accelerate the nonbonded computations, the remaining computations, left unaccelerated, would limit our maximum speedup to a factor of 10. Hence, we dedicated a significant fraction of silicon area to accelerating tasks such as bonded force computation, constraint computation, velocity and position updates, and atom migration, incorporating programmability as appropriate to accommodate a wide variety of force fields and integration methods.

Because the hardware used to perform integration is programmable, Anton also supports other simulation methodologies based on molecular mechanics force fields, such as Brownian dynamics and Monte Carlo simulation [14].

4. SYSTEM ARCHITECTURE

The building block of the system is a node, depicted in Figure 1. Each node comprises an MD-specific ASIC, attached DRAM, and six ports to the system-wide interconnection network. The nodes, which are logically identical, are connected in a three-dimensional torus topology (which maps naturally to the periodic boundary conditions described in Section 2). The initial version of Anton will be a 512-node torus with eight nodes in each dimension, as illustrated in Figure 2, but our architecture also supports larger and smaller toroidal configurations. For physical packaging, four nodes make up a node board, 32 node boards fit in a 19-inch rack, and four racks form a 512-node machine. A 512-node machine can handle chemical systems up to 200,000 particles using on-chip memory. For larger chemical systems, we split each box into virtual *sub-boxes*, and each node processes sub-boxes serially, paging sub-box state to the node's local DRAM; we refer to this as *DRAM mode*.

The remainder of this section details the architecture and design of the MD-specific ASIC, which has four major subsystems. The *high-throughput interaction subsystem* (HTIS) calculates explicit pairwise nonbonded interactions and performs charge spreading and force interpolation. The HTIS applies massive parallelism to these operations, which constitute the bulk of the calculation in MD. The *memory subsystem* drives the node's local DRAM and accumulates force terms. The *flexible subsystem* controls the ASIC and handles all other computations, including the bonded force calculations, the FFT, and integration. The *communication subsystem* provides both inter-chip and intra-chip communication. Between chips, each torus link provides 5.3 GB/s full-duplex communication with a hop latency around 50 ns. Within a chip, two 256-bit communication rings link all subsystems and the six inter-chip torus ports. The ASICs are clocked at a modest 400 MHz, with the exception of one double-clocked component in the HTIS (discussed in Section 4.1).

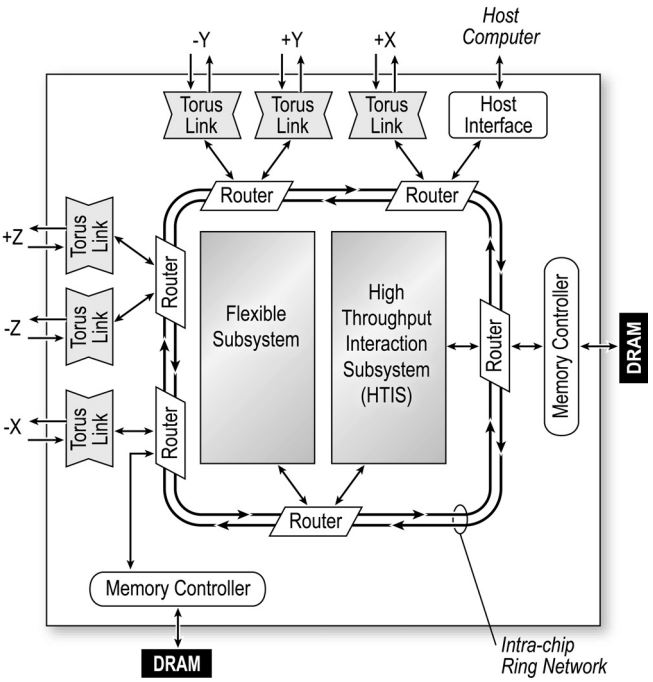

Figure 1. Processing node detail. The high-throughput interaction subsystem performs nonbonded MD interaction calculations. The flexible subsystem performs the remaining MD calculations, coordinates MD time step activity, and manages housekeeping tasks. Also shown are the intra-chip communication rings, the six connections to the inter-chip torus communication network, the host interface, and the two off-chip DRAM controllers.

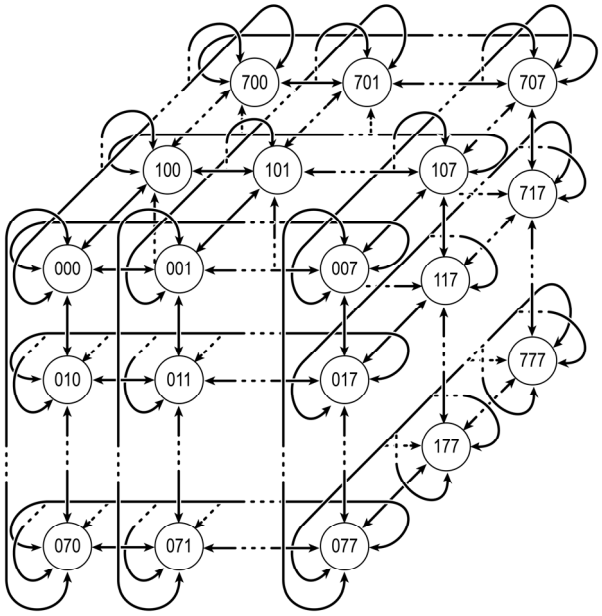

Figure 2. Three-dimensional torus interconnect topology. Circles represent processing nodes. Lines with arrows represent the links among the processing nodes. The XYZ notation for each processing node specifies the node's X, Y, and Z coordinates within the torus.

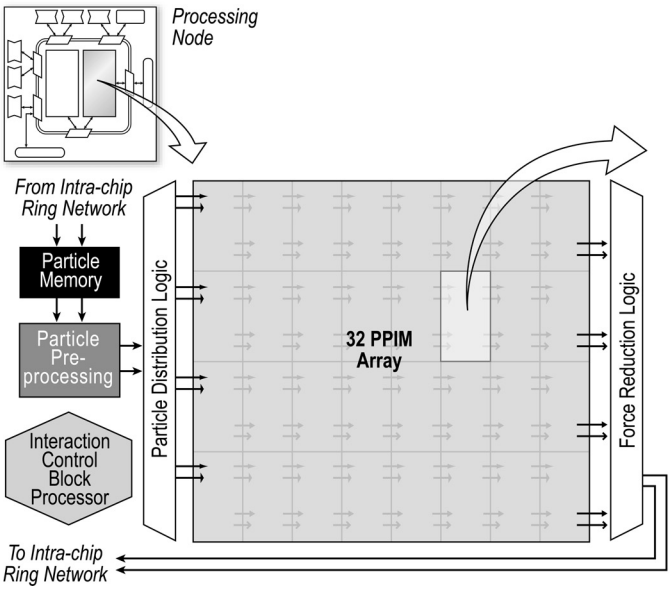

Figure 3. High-throughput interaction subsystem (HTIS) detail. The HTIS comprises an array of 32 Pairwise Point Interaction Modules (PPIMs) and an embedded control processor to coordinate the distribution of particle pairs to the PPIM array.

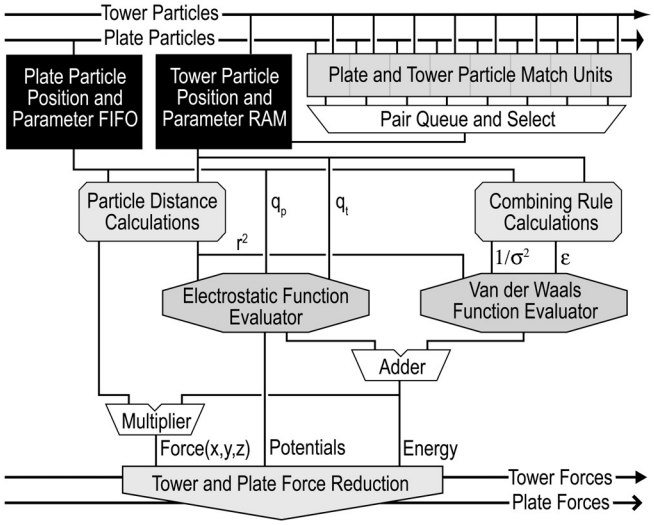

Figure 4. PPIM detail. This figure gives a sense of the numerical calculation units in PPIM. The top portion of the figure shows the matchmaking units and particle memories. The lower portion of the figure shows the general structure of the force, potential, and energy calculation pipelines.

4.1 High-Throughput Interaction Subsystem (HTIS)

The HTIS considers interactions between all points in one set (the tower) and all points in another set (the plate), computing nonzero interactions for pairs that are separated by less than a given cutoff radius and that satisfy certain other criteria, as prescribed by the NT parallelization method (see Section 2.3). For the explicit pairwise nonbonded calculations, both the tower and the plate consist of particles, and the HTIS computes and tabulates the forces that pairs of particles exert on one another.[1] For charge spreading, the HTIS interacts a tower of particles with a plate of grid points to determine how much of each particle's charge should be mapped to each neighboring grid point. For force interpolation, the HTIS also interacts a tower of particles with a plate of grid points, but uses the convolution value at each grid point to calculate the long-range force on each tower particle. Because we use the *k*-GSE method for long-range electrostatics, all three of these computations involve repeated evaluation of functions of distance between two points, permitting the same hardware pipelines to be used in all three cases. For expository simplicity, our description focuses on the operation of the HTIS as it computes the explicit pairwise nonbonded forces. The HTIS performs these calculations using an in-order, systolic architecture that accumulates forces on each particle as data streams through.

Figure 3 depicts the internal structure of the HTIS, which contains two major sub-blocks: an *Interaction Control Block* (ICB) and an array of 32 *Pairwise Point Interaction Modules* (PPIMs). The ICB comprises two communication ring interfaces, a large buffer area, and an embedded processor core. The ICB processor controls the flow of data through the HTIS and, through programmable ISA extensions, acts as a buffering, prefetching, synchronization, and writeback controller for the HTIS. The ICB either expects plate and tower particles to arrive via push communication or (in DRAM mode) reads plate and tower particles into its buffers. Then the ICB loads the tower particles into the PPIM array and streams the plate particles through that array, past the tower particles. Each force exiting the array is written to the memory system. After all plate particles have been processed, the ICB instructs the PPIM array to stream out the tower particle forces as well. The number of PPIMs was chosen on the basis of detailed system-level simulations with the goal of maximizing overall system performance, appropriately balancing the die area of the HTIS against that of the flexible subsystem.

The centerpiece of each PPIM is a force calculation pipeline that computes the force between a pair of particles; this is a 26-stage pipeline (at 800 MHz) of adders, multipliers, function evaluation units, and other specialized datapath elements. Inside this pipeline, we use customized numerical precisions: functional unit width varies across the different pipeline stages but still produces a sufficiently accurate 32-bit fixed-point result.

The various PPIM datapath widths were chosen to minimize die area and execution time while meeting a root mean squared force error criterion (see Section 5.3) for representative chemical systems, as determined in empirical studies with actual simulation data. These choices were then validated by running simulations for millions of time steps to ensure that precision limitations and rounding do not adversely affect simulation quality.

Feeding the force calculation pipeline are eight dedicated *matchmaking units* that collectively check each arriving plate

[1] In addition to forces acting on particles, both energy and electrostatic potential are also of interest. The HTIS can be configured to compute force, energy, or electrostatic potential. We focus on force calculations for the present discussion; the computations of energy and electrostatic potential are similar.

particle against all the tower particles stored in the PPIM to determine whether the plate particle is within a programmable cutoff radius from each of the tower particles. Each tower/plate particle pair that falls within the cutoff radius and satisfies certain other criteria goes to the PPIM's force calculation pipeline. As plate particles stream by, forces on tower particles are accumulated within the PPIM, while the accumulated force on each plate particle is streamed along with that particle. The PPIM, illustrated in Figure 4, runs at 800 MHz, while the rest of the ASIC runs at 400 MHz.

Because the HTIS streams plate particles past stored tower particles, it is simple to scale the PPIM array to any number of PPIMs. On the input side, wiring replicates particle records so that the same stream of plate particles flows through each row of the array. On the output side, force combiners merge streams of forces, adding up forces for each plate particle.

The PPIMs are the most hard-wired component of our architecture, reflecting the fact that they handle the most computationally intensive parts of the MD calculation. That said, even the PPIMs include programmability where we anticipate potential future changes to force fields. For instance, the functional forms for van der Waals and electrostatic interactions are specified using SRAM lookup tables, whose contents are determined at runtime.

4.2 Flexible Subsystem

The flexible subsystem handles a variety of tasks, some involving calculation and others involving system management and maintenance functions. It initiates each force computation phase by multicasting particle positions to the HTIS and flexible subsystems of multiple ASICs. It handles those parts of force computation not performed in the HTIS, including calculation of bonded force terms, the FFT, and the force correction terms. It performs all integration tasks, including constraint calculations, position and velocity updates, computation of global temperature and pressure, and atom migration. Lastly, it performs all maintenance activities, including boot, diagnostics, self-test, loading MD simulations, switching contexts, logging, checkpointing, and error reporting.[2]

Figure 5 shows the components of the flexible subsystem. Four identical *processing slices* form the core of the flexible subsystem. Each slice comprises a general-purpose core with its caches; a remote access unit (RAU) that performs autonomous data transfers; and two geometry cores (GCs), which are programmable cores whose ISA has been tailored to MD calculations. Other components of the flexible subsystem include a correction pipeline, which computes force correction terms; a racetrack, which serves as a local, internal interconnect for the flexible subsystem components; and a ring interface unit, which allows the flexible subsystem components to transfer packets to and from the communication subsystem.

The *general-purpose cores* manage the data transfers that occur during the time step and perform a few critical synchronizations. Each core implements a general-purpose integer and floating-point instruction set as well as some custom instructions to

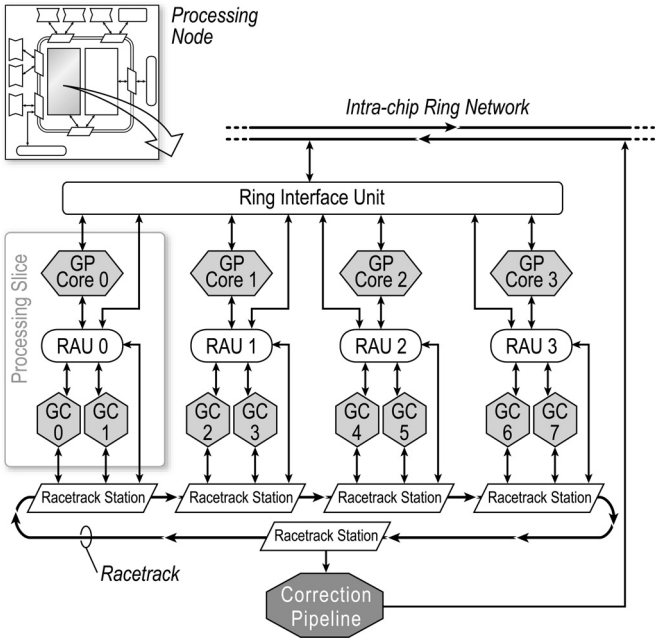

Figure 5. Flexible subsystem detail. The flexible subsystem is a collection of four identical processing slices (one of which is indicated by a box at the left) and a correction pipeline unit. The processing slices communicate with each other and with the correction pipeline via the racetrack. The various components communicate with the intra-chip communication ring via the ring interface unit shown at the top of the figure.

communicate at low latency with the other three general-purpose cores. In addition to the usual system interface and cache interfaces, each core also connects to a 32 KB scratchpad memory in the core's attached RAU. This scratchpad memory is used to stage MD simulation data for background transfer by the RAU. The general-purpose cores also handle all the maintenance tasks. These tasks, which are not performance-critical, occur either periodically (e.g., checkpointing every million time steps) or in response to explicit notification (e.g., an interrupt).

The *remote access unit* (RAU) is a programmable data transfer engine. In addition to a scratchpad memory that the associated general-purpose core can read and write, the RAU provides an array of transfer descriptors and associated state machines. A transfer descriptor describes a data transfer between the scratchpad memory and the rest of the system. Once the general-purpose core has initialized a transfer descriptor and marked it active (done with a single store instruction), the RAU takes over and performs the transfer itself, freeing the general-purpose core to perform other tasks. A transfer descriptor can also track push writes into the scratchpad memory, providing a fast polling and synchronization mechanism. The RAU implements 128 transfer descriptors, allowing multiple concurrent transfers. This background data transfer capability is crucial for performance, as it enables overlapped communication and computation.

The *geometry cores* (GCs) perform most of the flexible subsystem's computation. Each GC is a dual-issue, statically-scheduled SIMD processor with pipelined multiply-accumulate support. The GC's basic data type is a vector of four 32-bit fixed-

[2] Many of the maintenance activities are divided between the flexible subsystem and an external host computer system.

point values, and two independent SIMD operations on these vectors issue each cycle. The GC's instruction set includes element-wise vector operations (for example, vector addition); more complicated vector operations such as a dot product; and scalar operations that read and write arbitrary scalar components of the vector registers (essentially accessing the SIMD register file as a larger scalar register file). Such scalar operation support is atypical of SIMD processors but has proven useful in MD. For example, the bonded calculations (see Section 2.1) make heavy use of this feature.

All fixed point arithmetic in the machine (including that in the GCs) can be thought of as operating on numbers in the range $[-1,1)$. Addition is allowed to "wrap" in the natural way for twos-complement arithmetic. With the prescription that $-1 \cdot -1 = -1$, this range is closed under multiplication, but one must define a rule for rounding. Multiplication can either round to nearest with ties going to nearest even, or round toward negative infinity. In the vast majority of cases, the software uses round to nearest. In addition to "normal" multiplication, the GCs support a "multiply with shift" operation that allows one to compute $a \cdot b \cdot 2^k$ at the maximum available precision. Another instruction counts the leading sign bits. These instructions can be combined to emulate floating point, but in practice, most of the quantities handled by the GCs are in well-characterized, bounded ranges (e.g., bonds are between 1 and 3 Å in length) so there is no need for software to dynamically normalize fixed point values.

The *correction pipeline* (CP) uses a structure similar to the PPIM force calculation pipeline to compute force correction terms that fully or partially cancel out certain nonbonded interactions (see Section 2.1). The correction pipeline also accumulates bonded force terms before they are sent to the memory subsystem, reducing communication traffic.

The *racetrack* is a private, local, unidirectional ring interconnect among the flexible subsystem components, permitting these components to exchange data with low latency. The racetrack supports multicasting of particles to the geometry cores for bonded force calculations, communication of bonded forces to the correction pipeline, and efficient intra-node communication during FFT computation.

4.3 Communication Subsystem

The communication subsystem provides high-speed, low-latency communication both between ASICs and among the subsystems within an ASIC. Most routing is performed using a global 48-bit address space, with 16 bits of node identifier and 32 bits of address per node. In addition, multicast group addresses allow delivery of a single source packet to multiple destinations on multiple ASICs; multicast is used for efficient position sending, for neighbor synchronization, and for all-reduce operations. The communication subsystem also provides a source routing mode for topology discovery and confirmation. The network provides flow control, back-pressuring senders under oversubscription, and it provides class-based admission control with rate metering. Routing is dynamic, and a combination of virtual circuit numbering and adaptive routing, to be described in a future paper, ensures deadlock freedom using a small number of virtual channels. The communication subsystem also allows access to an external host computer system for input and output of simulation data.

4.4 Memory Subsystem

The memory subsystem provides access to the ASIC's attached DRAM. In addition to basic memory read/write access, the memory subsystem supports accumulation and synchronization. Special memory write operations numerically add incoming write data to the contents of the memory location specified in the operation. These operations implement force, energy, potential and spread charge accumulations, reducing the computation and communication load on the flexible subsystem. Atomic memory operations synchronize the ASIC and the external host computer system.

4.5 Parallel Operation

Anton is programmed using a combination of C, assembly language, and control of hardware engines through memory-mapped registers (MMRs). The general-purpose cores and the ICB processor have an optimizing C compiler; the geometry cores are programmed in assembly language. Other components, such as the correction pipeline, the routing tables of the communication subsystem, and the synchronization features of the memory subsystem, are configured through MMR accesses.

The HTIS, flexible, communication, and memory subsystems are independent hardware units, and operate in parallel. Force computation consists of three steps. First, the HTIS spreads charges for the long-range calculations while the flexible subsystem calculates most of the bonded force and correction force terms. Second, the HTIS calculates explicit pairwise nonbonded forces while the flexible subsystem calculates the FFT and inverse FFT. Third, the HTIS performs force interpolation while the flexible subsystem completes the remainder of the bonded force and correction force terms. With proper tuning, the explicit pairwise, bonded, and correction force terms are "hidden" behind the long-range work. Throughout all three steps, the memory subsystem receives force reduction packets and accumulates the net force on each particle. The HTIS is idle during the integration phase, and is disabled to reduce power. The flexible subsystem code, however, partially overlaps the accumulation of forces from the previous time step with integration and position broadcast for the next time step.

5. PERFORMANCE AND ACCURACY MEASUREMENTS

In this section, we show that the performance of Anton significantly exceeds that of other MD platforms, and that Anton is capable of performing simulations of high numerical accuracy.

5.1 Simulation Environment

Because we do not yet have working hardware, performance estimates for our machine come from our performance simulator. We wrote the simulator in C++, using a cycle-driven methodology and an infrastructure that allows us to mix and match C++ and Verilog RTL components for design, algorithm, and performance verification purposes. The cycle fidelity of our performance simulator varies across components; models of the communication subsystem routers as well as the flexible subsystem processor cores are cycle-accurate (i.e., the C++ and RTL simulations produce identical bits at identical times), while the models of the HTIS, RAU, and memory subsystem are within 10% of cycle

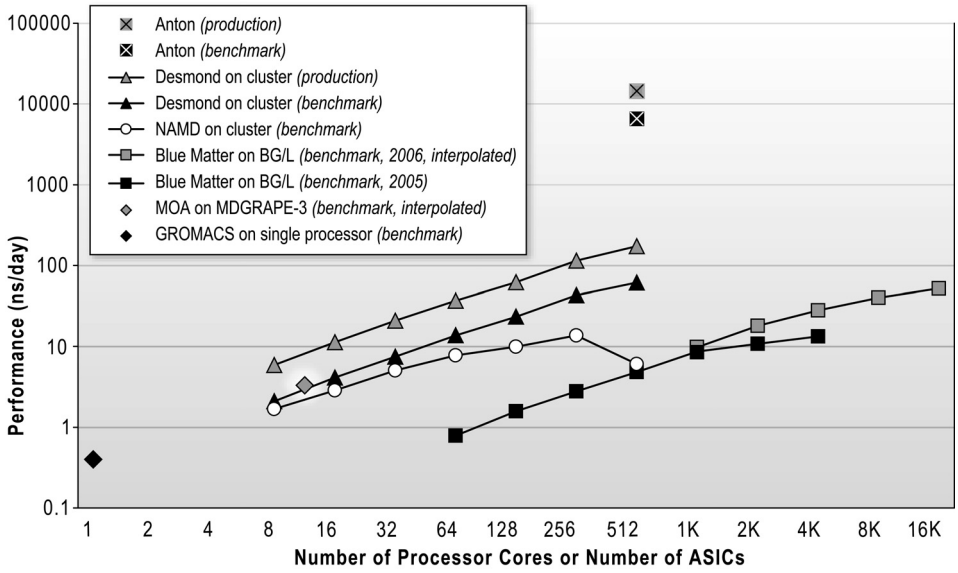

Figure 6. Performance comparison of MD platforms. The vertical axis shows performance in ns/day (higher is better); the horizontal axis shows the number of processor cores (for the cluster, Blue Gene/L, and GROMACS) or the number of ASICs (for Anton and MDGRAPE-3). Benchmark parameters were chosen to make the resulting simulation rates directly comparable, while production parameters were chosen to typify expected operation of Anton and Desmond.

accurate (i.e., the C++ and RTL simulations produce identical bits, but at times that vary within ±10%). A few analog sections of the design, such as the DRAM interface and the physical layer of our communication link, do not lend themselves to digital simulation; for these pieces we use time estimates generated by the hardware designers and expect fidelity within ±20%. The simulator currently comprises about 100,000 lines of C++ code and simulates a 512-node version of our machine at about ten machine clock cycles every second.

5.2 Performance Comparison to Other MD Platforms

We compare the performance of various MD platforms in terms of simulation rate (nanoseconds of simulated time per day of execution) on a particular chemical system. In this section and in 5.3, we use a system with 23,558 atoms in a cubic box measuring 62.2 Å on a side. This system represents dihydrofolate reductase (DHFR), a protein targeted by various cancer drugs, surrounded by water. The same chemical system is used in the Joint AMBER-CHARMM MD benchmark [1], and our simulations, like those of that benchmark, used the CHARMM force field [21].

We compared the simulation rate of DHFR on our machine to the rate on the following MD platforms:

- A state-of-the-art commodity cluster (consisting of Sun Fire V20z servers, each with two 2.4 GHz AMD Opteron Model 250 single-core processors, connected by an Infiniband network) running the following software packages:
  - NAMD, widely regarded as the fastest MD software for commodity clusters at high levels of parallelism [24].
  - GROMACS, widely regarded as the fastest MD software for single-processor runs [33].
  - Desmond, an MD software package for commodity clusters that we are developing [4]. Desmond significantly outperforms previously described MD codes at high levels of parallelism on commodity clusters, taking advantage of novel techniques we discovered while designing our machine.
- IBM's massively parallel, general-purpose Blue Gene/L machine, running its Blue Matter MD code, which was designed for maximal scalability [11, 12, 15].
- MDGRAPE-3, the most recent specialized ASIC for MD from the MDGRAPE project, running their MOA software[31].

Both the performance and the accuracy of an MD simulation depend on the settings of a number of simulation parameters such as the cutoff radius for explicit pairwise nonbonded force calculations, the choice of method for long-range electrostatics, and the length of the time step. Figure 6 shows measured or estimated performance for each MD platform using *benchmark parameters* chosen to be comparable to the parameters used by IBM in a set of performance results reported in 2005 for Blue Matter running the DHFR system on Blue Gene/L [11]. We also included a set of improved performance estimates for Blue Matter interpolated from performance results published recently for chemical systems other than DHFR [12]. We ran NAMD, GROMACS, and Desmond on our cluster using parameters identical to those used by IBM. Anton uses *k*-GSE rather than PME for long-range electrostatics, necessitating the choice of a different set of benchmark parameters for force computation; to ensure a fair comparison, we used the same integration parameters as Blue Matter and selected force calculation parameters for *k*-GSE that lead to more accurate computed forces than those used by Blue Matter (as shown in Section 5.3). Parameter settings for all runs are shown in Table 2.

We estimated MDGRAPE-3's performance on DHFR using the MDGRAPE-3 designers' theoretical performance model [31]. This model assumes direct summation for all particle pairs, which should show the best sustained performance [31], because

Table 2. Parameters used in timing runs on dihydrofolate reductase (DHFR) on various platforms. Benchmark parameters were chosen to make the resulting simulation rates directly comparable, while production parameters were chosen to typify expected operation of Anton and Desmond. Parameters that differ from those in the first row appear in bold face. All PME runs used fourth-order splines, while all *k*-GSE runs used 180 grid points of support for charge spreading and force interpolation.

Platform	Time Step	Constraints	Cutoff	Long-range Method	Long-range Frequency	FFT Mesh
Blue Matter on BG/L (benchmark, 2005) Blue Matter on BG/L (benchmark, 2006, interpolated) Desmond on cluster (benchmark) GROMACS on single processor (benchmark) NAMD on cluster (benchmark)	1fs	No	9Å	PME	1 step	64x64x64
MOA on MDGRAPE-3 (benchmark, interpolated)	1fs	No	**None**	**None**	NA	NA
Anton (benchmark)	1fs	No	**13**Å	***k*-GSE**	1 step	**32x32x32**
Desmond on cluster (production)	**2.5fs**	**Yes**	9Å	PME	**2 steps**	64x64x64
Anton (production)	**2.5fs**	**Yes**	**13**Å	***k*-GSE**	**2 steps**	**32x32x32**

MDGRAPE-3 does not include hardware acceleration for efficient long-range electrostatics methods.

We also measured the simulation rate of Anton and Desmond with *production parameters* representative of those we expect to use in actual operation. Force calculation settings are identical for benchmark and production parameters, but runs using production parameters employed a constrained integrator with longer time steps, in which long-range forces are evaluated only every other time step. We show in Section 5.3 that these runs satisfy common accuracy criteria, but rigorously comparing the accuracy of runs performed using constrained and unconstrained integrators is difficult, as these integrators assume slightly different physical models.

We expect a 512-node version of Anton to simulate 14,500 ns/day (1 ms in 69 days) on the DHFR system using production parameters, and 6,600 ns/day using benchmark parameters (Figure 6). This compares to a maximum benchmark parameter simulation rate of 14 ns/day for NAMD on our cluster (using 256 processor cores on 128 cluster nodes[3]). In other words, we expect Anton to exceed by a factor of nearly 500 the fastest simulation rate currently achievable using generally available codes on commodity hardware.

Desmond's maximum simulation rate on the same cluster is 173 ns/day with production parameters and 62 ns/day with benchmark parameters (in both cases using 512 processor cores on 256 cluster nodes). In other words, we expect Anton to be 80–100 times faster than Desmond is today. Desmond's performance advantage over other currently available MD codes is largely attributable to techniques we developed while designing Anton; the adaptability of these techniques to commodity hardware is an additional benefit of our work.

The fastest published simulation rate for Blue Matter running DHFR with benchmark parameters is 13 ns/day, using 4,096 processor cores (2,048 nodes) of Blue Gene/L [11]. IBM has since reported improved performance and scalability for Blue Matter; by interpolation of performance results published recently for smaller and larger systems [12], we estimated a simulation rate for DHFR of 40 ns/day on 8,192 processor cores (4,096 nodes) of Blue Gene/L and 50 ns/day on 16,384 processor cores (8,192 nodes). An even more recent report says that DHFR runs at somewhat more than 60 ns/day on 8,192 processor cores (4,096 nodes) of Blue Gene/L (R. S. Germain, personal communication). We expect a 512-node version of Anton to exceed this simulation rate by approximately a factor of 100.

The performance of other MD platforms will undoubtedly improve by 2008, when Anton is scheduled for completion and use in biological research, but Anton's performance advantage over current MD platforms significantly exceeds the speedup predicted by Moore's law over that period.

5.3 Accuracy

Unfortunately, no single metric adequately captures the accuracy of an MD simulation. This section applies two common accuracy measures to simulations by Anton.

To quantify the error in force computation, we measured the *relative rms force error*, defined as the root mean squared error in the force on all particles divided by the root mean squared force [28]. Determination of force error is possible because, for a given force field and a given set of particle coordinates, we can closely approximate the "true" force specified by the force field on each particle by evaluating the force field equations very accurately. We then compare these "true" forces to those computed by our machine with a given set of force calculation parameters, using a numerical emulator that exactly duplicates Anton's limited-precision, fixed-point arithmetic. The relative rms force error for our machine using either its benchmark parameters or its production parameters (which lead to identical computed forces), as measured on the DHFR system, is 1.5×10^{-4}. For comparison, the force calculation benchmark parameters specified by the Joint AMBER-CHARMM benchmark and used in the Blue Matter performance measurements reported in Section 5.2 give a higher relative rms force error of 3.0×10^{-4} even when all computation is performed in double-precision floating-point arithmetic. A relative rms force error below 10^{-3} is generally considered sufficiently accurate for biomolecular MD simulations [35, 36].

To measure the overall accuracy of our production runs, we also measured *energy drift*. An exact MD simulation would conserve energy exactly. Errors in the simulation generally lead to

[3] In our hands, NAMD actually simulated DHFR more *slowly* on 512 processor cores than on 256. We used NAMD version 2.6b1 for these experiments; more recent versions of NAMD incorporate performance improvements (J. Phillips, personal communication).

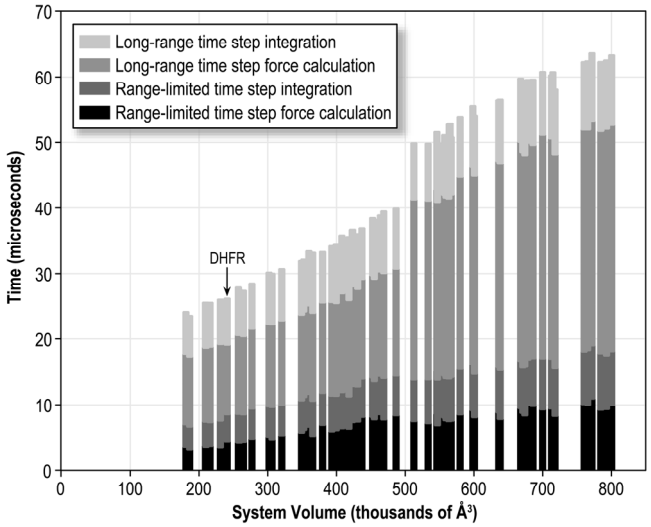

Figure 7. Scaling of performance for a 512-node version of Anton with increasing chemical system size. The graph shows a stacked bar chart for each chemical system, with the height of each stack proportional to the simulation time, assuming that long-range forces are evaluated every other time step. Each stack represents the time required to execute two consecutive time steps; one is a "long-range time step" that includes calculation of long-range electrostatics by *k*-GSE, and the other is a "range-limited time step" that does not. The chemical systems represent proteins and nucleic acids of various sizes, surrounded by water.

an increase in the overall energy of the simulated system with time, a phenomenon known as energy drift. We used the numerical emulator of our machine to integrate the DHFR system with production parameters over 5 ns of simulated time (2 million time steps). While the total energy exhibited short-term fluctuations of a few kcal/mol (about 0.001% of the total system energy of 350,000 kcal/mol), there was no detectable long-term trend in total energy. MD studies are generally considered more than adequate even with a significantly higher energy drift [34]. It is worth noting, however, that Blue Matter has achieved an extremely low level of energy drift [12], which will probably not be matched by Anton.

5.4 Scaling with Chemical System Size

Within the range where chemical systems fit in on-chip memory, we expect performance to scale roughly linearly with the number of atoms, albeit with occasional jumps as different operating parameters change to optimize performance while maintaining accuracy. Figure 7 shows the scaling of performance with chemical system size. The largest discontinuity in simulation rate occurs at a system volume of approximately 500,000 $Å^3$ when we change from a 32×32×32 FFT grid to a 64×64×64 FFT grid, reflecting the fact that our code supports only power-of-two-length FFTs. This lengthens the long-range calculation because the number of grid points increases by a factor of 8. Overall, the results are consistent with supercomputer scaleup studies—as we increase chemical system size, our efficiency improves, because we are better able to overlap communication with computation and because our pipelines operate closer to peak efficiency.

6. CONCLUSION

We have designed, and are currently in the process of implementing, a specialized, massively parallel machine, called Anton, for the high-speed execution of molecular dynamics simulations. We expect that Anton will be capable of simulating the dynamic, atomic-level behavior of proteins and other biological macromolecules in an explicitly represented solvent environment for periods on the order of a millisecond—about three orders of magnitude beyond the reach of current molecular dynamics simulations. The machine is being implemented using specialized ASICs, each of which performs a very large number of application-specific calculations during each clock cycle. Novel architectural and algorithmic techniques are used to minimize intra- and inter-chip communication, providing an unusually high degree of scalability.

While it contains programmable elements that could in principle support the parallel execution of algorithms for a wide range of other applications, Anton was not designed to function as a general-purpose scientific supercomputer, and would not in practice be well suited for such a role. Rather, we envision Anton serving as a "computational microscope," allowing researchers to observe for the first time a wide range of biologically important structures and processes that have thus far proven inaccessible to both computational modeling and laboratory experiments. To the extent that we are successful in achieving our research objectives, we would hope that Anton might make significant contributions to both the advancement of basic scientific knowledge and the development of safe, effective, precisely targeted drugs capable of relieving suffering and saving human lives.

7. REFERENCES

[1] MD Benchmarks for Amber, CHARMM and NAMD, See http://amber.scripps.edu/amber8.bench2.html.

[2] S. A. Adcock and J. A. McCammon, Molecular Dynamics: Survey of Methods for Simulating the Activity of Proteins, *Chem. Rev.*, 106: 1589-1615, 2006.

[3] J. Banerjee, D. K. Hsiao, and R. I. Baum, Concepts and Capabilities of a Database Computer, *ACM Transactions on Database Systems*, 3(4): 347-384, 1978.

[4] K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, and D. E. Shaw, Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters, *Proc. ACM/IEEE Conf. on Supercomputing (SC06)*, Tampa, FL, 2006.

[5] K. J. Bowers, R. O. Dror, and D. E. Shaw, The Midpoint Method for Parallelization of Particle Simulations, *J. Chem. Phys.*, 124: 184109, 2006.

[6] K. J. Bowers, R. O. Dror, and D. E. Shaw, Zonal Methods for the Parallel Execution of Range-Limited N-Body Problems, *J. Comput. Phys.*, 221(1):303-329, 2007.

[7] C. L. Brooks, B. M. Pettit, and M. Karplus, Structural and Energetic Effects of Truncating Long Ranged Interactions in Ionic and Polar Fluids, *J. Chem. Phys.*, 83(11): 5897-5908, 1985.

[8] I. Brooks, C.L. and D. A. Case, Simulations of Peptide Conformational Dynamics and Thermodynamics, *Chem. Rev.*, 93: 2487-2502, 1993.

[9] Y. Duan and P. A. Kollman, Pathways to a Protein Folding Intermediate Observed in a 1-Microsecond Simulation in Aqueous Solution, *Science*, 282(5389): 740-744, 1998.

[10] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen, A Smooth Particle Mesh Ewald Method, *J. Chem. Phys.*, 103(19): 8577-8593, 1995.

[11] B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. Giampapa, Y. Zhestkov, M. C. Pitman, F. Suits, A. Grossfield, J. Pitera, W. Swope, R. Zhou, S. Feller, and R. S. Germain, Blue Matter: Strong scaling of Molecular Dynamics on Blue Gene/L, *Proc. International Conf. on Computational Science (ICCS 2006)*, V. Alexandrov, D. van Albada, P. Sloot, and J. Dongarra, Eds., Springer-Verlag, *LNCS*, 3992:846-854, 2006.

[12] B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. E. Giampapa, M. C. Pitman, and R. S. Germain, Blue Matter: Approaching the Limits of Concurrency for Classical Molecular Dynamics, *Proc. ACM/IEEE Conf. on Supercomputing (SC06)*, Tampa, FL, 2006.

[13] R. D. Fine, G. Dimmler, and C. Levinthal, FASTRUN: A Special Purpose, Hardwired Computer for Molecular Simulation, *Proteins: Struct, Funct, Genet*, 11(4): 242-253, 1991 (erratum: 14(3): 421-422, 1992).

[14] D. Frenkel and B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*, Second ed. London: Academic Press, 2001.

[15] R. S. Germain, B. Fitch, A. Rayshubskiy, M. Eleftheriou, M. C. Pitman, F. Suits, M. Giampapa, and T. J. C. Ward, Blue Matter on Blue Gene/L: Massively Parallel Computation for Biomolecular Simulation, *Proc. 3rd IEEE/ACM/IFIP International Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS '05)*, 207–212, New York, NY, 2005.

[16] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*. Bristol: Adam Hilger, 1988.

[17] W. L. Jorgensen, D. S. Maxwell, and J. Tirado-Rives, Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids, *J. Am. Chem. Soc.*, 118(45): 11225-11236, 1996.

[18] M. Karplus and J. A. McCammon, Molecular Dynamics Simulations of Biomolecules, *Nat. Struct. Bio.*, 9(9): 646 - 652, 2002.

[19] P. A. Kollman, R. W. Dixon, W. D. Cornell, T. Fox, C. Chipot, and A. Pohorille, The Development/Application of a "Minimalist" Organic/ Biomolecular Mechanic Forcefield Using a Combination of Ab Initio Calculations and Experimental Data, in *Computer Simulation of Biomolecular Systems: Theoretical and Experimental Applications*, W. F. van Gunsteren and P. K. Weiner, Eds. Dordrecht, Netherlands: ESCOM, 1997, 83-96.

[20] D. K. Layer and C. Richardson, Lisp Systems in the 1990s, *Communications of the ACM*, 34(9): 48-57, 1991.

[21] J. MacKerell, D. Bashford, M. Bellott, R. L. Dunbrack, J. D. Evanseck, M. J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph-McCarthy, L. Kuchnir, K. Kuczera, F. T. K. Lau, C. Mattos, S. Michnick, T. Ngo, D. T. Nguyen, B. Prodhom, I. Reiher, W. E., B. Roux, M. Schlenkrich, J. C. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiórkiewicz-Kuczera, D. Yin, and M. J. Karplus, All-Atom Empirical Potential for Molecular Modeling and Dynamics Studies of Proteins, *J. Phys. Chem. B*, 102(18): 3586-3616, 1998.

[22] P. Mark and L. Nilsson, Structure and Dynamics of Liquid Water with Different Long-Range Interaction Truncation and Temperature Control Methods in Molecular Dynamics Simulations, *J. Comput. Chem.*, 23(13): 1211-1219, 2002.

[23] V. S. Pande, I. Baker, J. Chapman, S. P. Elmer, S. Khaliq, S. M. Larson, Y. M. Rhee, M. R. Shirts, C. D. Snow, E. J. Sorin, and B. Zagrovic, Atomistic Protein Folding Simulations on the Submillisecond Time Scale Using Worldwide Distributed Computing, *Biopolymers*, 68(1): 91-109, 2003.

[24] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, Scalable Molecular Dynamics with NAMD, *J. Comput. Chem.*, 26(16): 1781-1802, 2005.

[25] S. Plimpton, Fast Parallel Algorithms for Short-Range Molecular-Dynamics, *J. Comput. Phys.*, 117(1): 1-19, 1995.

[26] T. Schlick, R. D. Skeel, A. T. Brunger, L. V. Kalé, J. A. Board, J. Hermans, and K. Schulten, Algorithmic Challenges in Computational Molecular Biophysics, *J. Comput. Phys.*, 151(1): 9-48, 1999.

[27] M. M. Seibert, A. Patriksson, B. Hess, and D. van der Spoel, Reproducible Polypeptide Folding and Structure Prediction Using Molecular Dynamics Simulations, *J. Mol. Biol.*, 354(1): 173-183, 2005.

[28] Y. Shan, J. L. Klepeis, M. P. Eastwood, R. O. Dror, and D. E. Shaw, Gaussian Split Ewald: A Fast Ewald Mesh Method for Molecular Simulation, *J. Chem. Phys.*, 122: 054101, 2005.

[29] D. E. Shaw, A Fast, Scalable Method for the Parallel Evaluation of Distance-Limited Pairwise Particle Interactions, *J. Comput. Chem.*, 26(13): 1318-1328, 2005.

[30] M. Snir, A Note on N-Body Computations with Cutoffs, *Theor. Comput. Syst.*, 37: 295-318, 2004.

[31] M. Taiji, T. Narumi, Y. Ohno, N. Futatsugi, A. Suenaga, N. Takada, A. Konagaya, Protein Explorer: A Petaflops Special-Purpose Computer System for Molecular Dynamics Simulations, *Proc. ACM/IEEE Conf. on Supercomputing (SC03)*, Phoenix, AZ, 2003.

[32] S. Toyoda, H. Miyagawa, K. Kitamura, T. Amisaki, E. Hashimoto, H. Ikeda, A. Kusumi, and N. Miyakawa, Development of MD Engine: High-Speed Accelerator with Parallel Processor Design for Molecular Dynamics Simulations, *J. Comput. Chem.*, 20(2): 185-199, 1999.

[33] D. van der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. C. Berendsen, GROMACS: Fast, Flexible, and Free, *J. Comput. Chem.*, 26(16): 1701-1718, 2005.

[34] W. Wang and R. D. Skeel, Fast Evaluation of Polarizable Forces, *J. Chem. Phys.*, 123(16): 164107, 2005.

[35] R. Zhou and B. J. Berne, A New Molecular Dynamics Method Combining the Reference System Propagator Algorithm with a Fast Multipole Method for Simulating Proteins and Other Complex Systems, *J. Chem. Phys.*, 103(21): 9444-9459, 1995.

[36] R. Zhou, E. Harder, H. Xu, and B. J. Berne, Efficient Multiple Time Step Method for Use with Ewald and Particle Mesh Ewald for Large Biomolecular Systems, *J. Chem. Phys.*, 115(5): 2348-2358, 2001.